EECS 183 Winter 2020

Exam 2

Closed Book 3" x 5" notecard No Electronic Devices Closed Neighbor

All cell phones and other electronic devices, including smart watches, must be turned off and placed in your backpack

Instructions

1. After we indicate you may start, print your uniqname at the top of every page of this booklet.
2. After we indicate you may start the exam, verify that you have received every page of the exam. There are 23 pages in this booklet, including this cover page.
3. You may have and reference one 3" x 5" notecard.
4. Assume all code and code fragments are syntactically valid, unless otherwise specified.
5. Assume/use only standard C++11.
6. In all the given code, if any character looks like a space, it is one.
7. Write clearly. If we cannot read your writing, it will be marked wrong. This includes indentation and curly braces.
8. Sign below and print your name and UMID. 10 pts off if we cannot read your name or UMID.

"I have neither given nor received aid on this examination, nor have I concealed any violations of the Honor Code."

_____

SIGN your name above

_____

Clearly and legibly print your **uniqname**

_____

Clearly and legibly print your **full name (NOT uniqname)** above

_____

Clearly and legibly print your **UMID** above

Problem Number	Points Possible
Multiple Choice	90
Free Response 1	13
Free Response 2	22
Free Response 3	7
Free Response 4	13
Free Response 5	20
Free Response 6	20
	185

You can use the area below for working out problems.

Multiple Choice [4.5 points each]

Record your choices for all multiple choice questions using the circles next to each question. Fill the single circle corresponding to your answer completely, like this: Ⓐ● Ⓒ Ⓓ Ⓔ

1. Which of the following about C++ classes is **true**?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. Classes, like arrays, can only have members of a single data type.
B. Classes include both data and functionality.
C. You cannot declare an array that holds elements of a class type.
D. All member functions must be `public`.
E. All of A, B, C, D are true.

2. When executing the following code, what is the first value of `index` that will cause an out of bounds access of array `arr`?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

```
const int SIZE = 5;
char arr[SIZE][SIZE];
for (int index = 0; index < SIZE; index++) {
    arr[index][index + 1] = index;
}
```

A. When `index == -1`
B. When `index == 0`
C. When `index == 4`
D. When `index == 5`
E. The code never goes out of bounds off the array.

3. Which of the following is NOT *a* stream *state*?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. good
B. bad
C. fail
D. eof
E. clear

4. Which of the following is NOT necessary when *writing* to a file in C++?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. Include the `fstream` library
B. Declare an instance of `ifstream` ofstream.
C. Open the file
D. Write to the file
E. All of the above are necessary steps for writing to a file in C++.

Consider the following function, which will be **used in the next two questions**.

For reference, English alphabet: ABCDEFGHIJKLM NOPQRSTUVWXYZ

The EECS 183 staff have written a new cipher function in C++, named `sillyCipher`. It is implemented in the following function:

```
void sillyCipher(const string &original, string &result) {
    for (int x = 0; x < original.size(); x++) {
        if (original.at(x) >= 'A' && original.at(x) <= 'M') {
            result.at(x) = 'A';
        }
        else if (original.at(x) >= 'N' && original.at(x) <= 'Z') {
            result.at(x) = 'B';
        }
    }
    return;
}
```

5.1. What would be stored in `res` by the end of `sillyCipher`'s execution by the following code fragment?
`string orig = "LISA";`
`string res = "NONE";`
`sillyCipher(orig, res);`
	(A) (B) (C) (D) (E)

A. `"AABA"`
B. `"ABAA"`
C. `"ABAB"`
D. `"BBAB"`
E. `"NONE"`

5.2. Which of the following sentences best describes the behavior of `sillyCipher`?	(A) (B) (C) (D) (E)

A. It will result in a runtime error if `original.size()` > `result.size()`.
B. It will not compile because `string` datatype cannot be passed by reference.
C. It will not work correctly because it has an off-by-one error.
D. It will not work correctly because one of the parameters is designated `const`.

Consider the following code fragment, which will be **used in the next three questions**.

```
ifstream infile;
infile.open("data.txt");
int total = 0;
int num;
while (infile >> num) {
    total += num;
    // add code for question 6.2 here
}
cout << total;
```

6.1. What does the above code print if `data.txt` is opened successfully and contains: `3 9 4.2 6`	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. 12
B. 16
C. 22
D. 24
E. The program goes into an infinite loop

6.2. Now, inside the `while` loop the following statement is added: `infile.clear();` What does the above code print if `data.txt` is opened successfully and contains: `3 9 4.2 6`	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. 12
B. 16
C. 22
D. 24
E. The program goes into an infinite loop

6.3. In the previous question, #6.2, what does the function `clear()` in the statement `infile.clear();` do?	Ⓐ Ⓑ Ⓒ Ⓓ

A. Removes the non-integer characters from the input stream.
B. Removes the failed state of the stream, putting the stream into a good state.
C. Both A and B
D. None of A, B, or C

7. Given the following function declaration:
 	
`int foo(int &input);`
 	
and the following variable declaration:
 	
`int number = 5;`
 	
Which of the following is/are valid function call(s)?
	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. `foo(5);`
B. `foo(number);`
C. `foo(foo(number));`
D. `foo(number + number);`
E. More than one of A, B, C, or D are valid function calls.

Consider the following function, which will be **used in the next two questions**.

```
int bar(int arr[], int size) {
    for (int i = 1; i < size; i++) {
        arr[i] += arr[i - 1];
    }
    return arr[size - 1];
}
```

8.1. For the given definition of bar, which of the following would be added to the Modifies of the RME for the function?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. cout
B. size
C. arr
D. i
E. More than one of A, B, C, or D

8.2. What prints when executing the following code fragment?
 	
`int test[5] = {1, 2, 3, 4, 5};`
`cout << bar(test, 4) << endl;`
	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. 10
B. 15
C. 7
D. 1
E. Nothing will print due to an off-by-one error in the function.

9. In C++, a class member function with the same name as the class would be described as which of the following?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. Instance
B. Class
C. Member variable
D. Constructor
E. Getter

Refer to the following ***buggy*** function definition which will be **used in the next two questions**.

```
/**
 * Requires: Nothing
 * Modifies: Nothing
 * Effects:  Returns a copy of original string with all alphabetical
 *           characters converted to lowercase. All other characters
 *           (numbers, symbols, spaces, punctuation marks, etc.) are
 *           unchanged.

 */
string toLowerCase(string original) {
    string copy;
    for (int i = 0; i < original.size(); i++) {
        copy[i] = tolower(original[i]);
    }
    return copy;
}
```

10.1. Which of the following tests will reveal the bug in the function?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. `cout << toLowerCase("hello");`
B. `cout << toLowerCase("Hello");`
C. `cout << toLowerCase("HELLO");`
D. All of A, B, and C will expose the bug
E. None of A, B, or C will expose the bug

10.2. Which of the following bugs exist in the function?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. Off-by-one error in the `for` loop.
B. Invalid index used when accessing a character of a string.
C. Incorrect return type
D. Cannot use brackets to access an individual character of a string
E. More than one of A, B, C, or D.

Consider the following class definition which will be **used in the next five questions**.

```
const int STACK_MAX = 100;

class Stack {
private:
    string lines[STACK_MAX];
    int stackPointer;

public:
    Stack();
    bool empty();
    int size();
    string peek();
    bool push(string item);
    string pop(int numElements);
};
```

11.1. Which of the following code snippets, in `main`, would create an instance of `Stack`?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. `Stack pile(STACK_MAX);`
B. `Stack pile();`
C. `Stack;`
D. `Stack pile;`
E. More than one of A, B, C, D would create an instance of `Stack`.

11.2. Consider the following variable declaration: `Stack myStack;` Which of the following is a valid call to the `empty` function?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. `bool isEmpty = myStack.empty();`
B. `bool isEmpty = myStack.empty;`
C. `empty(myStack);`
D. `cout << empty(myStack);`
E. `bool isEmpty = Stack::empty();`

11.3. Which of the following creates an array of `Stack` instances named `myStacks`?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. `Stack[10];`
B. `Stack myStacks;`
C. `Stack myStacks[10];`
D. `Stack[10];`
E. `Stack myStacks = Stack(10);`

11.4. Consider the array declaration of type `Stack` from the previous problem. How would you print the size of the last element of the array `myStacks`?	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. `cout << myStacks[9].stackPointer;`
B. `cout << myStacks[9].size();`
C. `cout << myStacks[9].size;`
D. `cout << myStacks.size();`
E. None of A, B, C, or D will print the `size` of the last element of the array `myStacks`.

11.5. Which of the following is a valid way to call the member function `push` given the following declaration?
 	
`Stack newStack;`
`string value = "Hello World!";`
	Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A. `value.push();`
B. `newStack.push();`
C. `value.push(newStack);`
D. `cout << newStack.push(value);`
E. `cout << newStack.push();`

12. Free Response 1 [13 points]

Implement the following function, which returns the index of an element in an array.
Example: The following code will print: `1 -1`

```
int test[] = { 1, 2, 3, 2, 4 };
cout << indexOf(test, 5, 2) << " " << indexOf(test, 5, 5);
```

```
/**
* Requires: size > 0, and size is the number of elements of arr
* Modifies: Nothing.
* Effects:  Returns the index of the first occurrence of value in
*           arr, or -1 if value is not in arr.
*/
int indexOf(int arr[], int size, int value) {
```

```
for(int i=0; i<size; i++){
  if(arr[i]==value){
    return i;
  }
  else{
    return -1;
  }
```

```
}
```

13. Free Response 2 [22 points]

A lower triangular matrix is a square matrix where every element that is strictly above the diagonal is equal to zero. For example, the matrices on the left are **not** lower triangular, while **both matrices on the right are lower triangular**.

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 & 0 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

The function below creates a lower triangular matrix `toMatrix` by copying the elements that are on or below the diagonal in `fromMatrix` to `toMatrix`, and setting the remaining elements of `toMatrix` to zero (within `size` bounds). For example, if `fromMatrix` is the matrix on the left above, `toMatrix` will be set to the matrix to the right. Fill in the function definition, assuming that matrices are stored with the top row first and the bottom row last.

```
/**
*  Requires: size > 0 and size <= MAX_SIZE, and size is the number
*            of rows and columns in both fromMatrix and toMatrix
*  Modifies: toMatrix
*  Effects : Copies the elements in and below the diagonal in
*            fromMatrix to toMatrix. Sets the remaining elements
*            in toMatrix to zero. Does not modify any elements that
*            are outside the size x size portion of toMatrix.
*/
void copyLowerTriangular(const int fromMatrix[MAX_SIZE][MAX_SIZE],
                         int toMatrix[MAX_SIZE][MAX_SIZE], int size) {
```

```
for(int i=0; i< size; i++){
  for(int j=0 ; j< size ;j++){
    if(i<j){
      toMatrix[i][j] = 0 ;
    }
    else{
      toMatrix[i][j] = fromMatrix[i][j] ;
    }
  }
}
```

}

The following class definition in the `Stack.h` file is relevant to the remaining questions on the exam.

```
const int STACK_MAX = 100;

class Stack {
private:
    // the stack is an array of strings
    string lines[STACK_MAX];

    /* stackPointer should always be the index of the element on "top"
    * of the stack.
    * The "top" of the stack is the element with the *greatest* index.
    * The value of stackPointer will be -1 when
    * the stack is empty, i.e., has 0 elements.
    *
    * Example:
    *  with 0 elements pushed on the stack, stackPointer = -1
    *  with 1 element pushed on the stack, stackPointer = 0
    *  with 2 elements pushed on the stack, stackPointer = 1
    *  with 3 elements pushed on the stack, stackPointer = 2
    *  and so on
    */
    int stackPointer;

public:
    /**
    * Requires: Nothing
    * Modifies: stackPointer
    * Effects:  Creates an empty stack
    */
    Stack() {
        stackPointer = -1;
    }

    /**
    * Requires: Nothing
    * Modifies: Nothing
    * Effects:  Returns the number of elements currently on the stack.
    */
    int size() {
        return stackPointer + 1;
    }
```

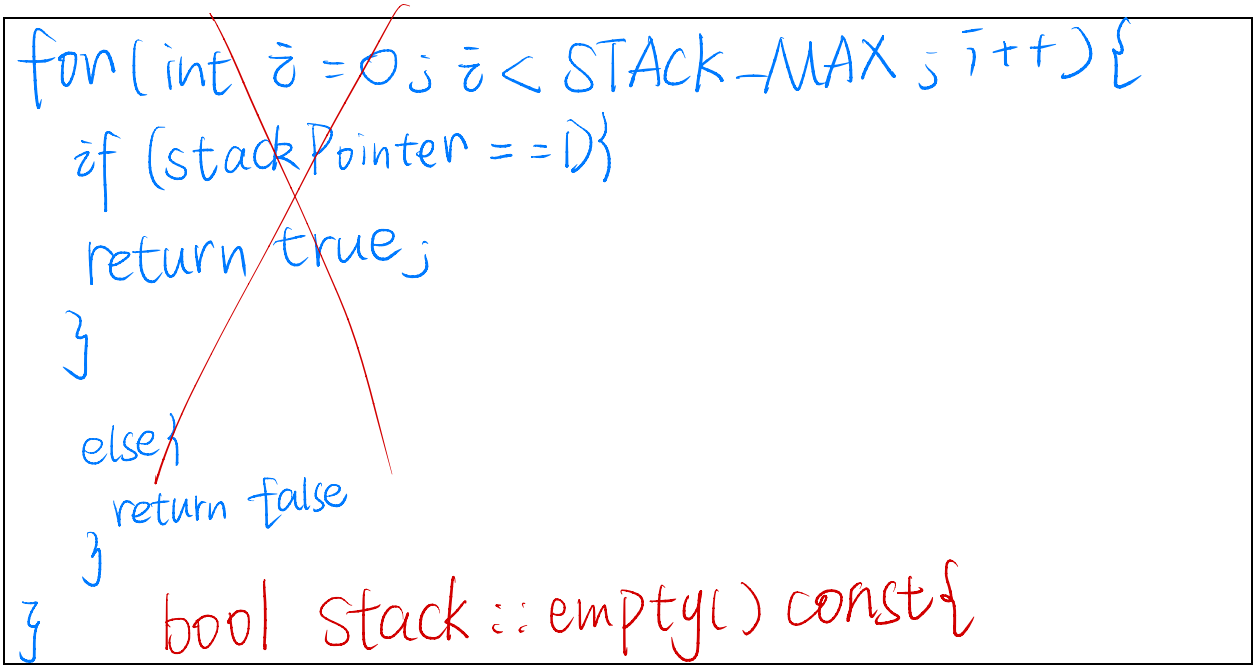
```
/**
* Requires: Nothing
* Modifies: Nothing
* Effects:  If stack is empty, returns true.
*           Otherwise returns false.
*/
bool empty();

/**
* Requires: Nothing
* Modifies: Nothing
* Effects:  If stack is not empty, returns the element
*           on top of the stack.
*           Otherwise returns an empty string.
*
* NOTE: Your implementation MUST use the member function empty()
*/
string peek();

/**
* Requires: Nothing
* Modifies: lines, stackPointer
* Effects:  Adds item on the top of the stack (lines)
*           if and only if there is space for it.
*           Returns true if item was added to lines,
*           returns false otherwise.
*
* NOTE: Your implementation MUST use the member function size()
*/
bool push(string item);
```

```
    /**
    * Requires: numElements >= 0
    * Modifies: stackPointer
    * Effects:  If stack is not empty, returns the element
    *           on top of the stack.
    *           Otherwise returns an empty string.
    *           Removes the number of elements specified by the
    *           parameter, or the number of elements in the stack,
    *           whichever is fewer.
    *           Removing one element from the stack is
    *           accomplished by decrementing the stack pointer by one.
    *
    * NOTE: Element returned is value on "top" before
    *       removing elements
    * NOTE: Your implementation MUST use the member function peek()
    */
    string pop(int numElements);
};
```

14.1. Free Response 3.1 [7 points]

Implement the `Stack` class member function **`empty`** below, **just as it would appear in the file `Stack.cpp`.** For your convenience, the function declaration from the `Stack` class definition is repeated below.

```
/**
 * Requires: Nothing
 * Modifies: Nothing
 * Effects:  If stack is empty, returns true.
 *           Otherwise returns false.
 */
bool empty();
```



14.2. Free Response 3.2 [13 points]

Implement the `Stack` class member function **peek** below, **just as it would appear in the file `Stack.cpp`.** For your convenience, the function declaration from the `Stack` class definition is repeated below.

```
/**
* Requires: Nothing
* Modifies: Nothing
* Effects:  If stack is not empty, returns the element
*           on top of the stack.
*           Otherwise returns an empty string.
*
* NOTE: Your implementation MUST use the member function empty()
*/
string peek();
```

14.3. Free Response 3.3 [20 points]

Implement the `Stack` class member function **`push`** below, **just as it would appear in the file `Stack.cpp`.** For your convenience, the function declaration from the `Stack` class definition is repeated below.

```
/**
* Requires: Nothing
* Modifies: lines, stackPointer
* Effects:  Adds item on the top of the stack (lines)
*           if and only if there is space for it.
*           Returns true if item was added to lines,
*           returns false otherwise.
*
* NOTE: Your implementation MUST use the member function size()
*/
bool push(string item);
```

14.4. Free Response 3.4 [20 points]

Implement the `Stack` class member function **pop** below, **just as it would appear in the file `Stack.cpp`.** For your convenience, the function declaration from the `Stack` class definition is repeated below.

```
/**
    * Requires: numElements >= 0
    * Modifies: stackPointer
    * Effects:  If stack is not empty, returns the element
    *           on top of the stack.
    *           Otherwise returns an empty string.
    *           Removes the number of elements specified by the
    *           parameter, or the number of elements in the stack,
    *           whichever is fewer.
    *           Removing one element from the stack is
    *           accomplished by decrementing the stack pointer by one.
    *
    * NOTE: Element returned is value on "top" before
    *       removing elements
    * NOTE: Your implementation MUST use the member function peek()
    */
    string pop(int numElements);
```